AD A106184

DTIC
S D

TECHNICAL REPORT

COMPUTER SCIENCE

12

Merging on Parallel Models of Computation

A. Borodin *
J.E. Hopcroft*

October 1981

TR ~~80~~ 81-472

Department of Computer Science
Cornell University
Ithaca, New York 14853

* University of Toronto
** Cornell University

This research was supported in part by ONR contract N00014-76-C-0018.

Merging on Parallel Models of Computation

A. Borodin*

J.E. Hopcroft**

Abstract

A variety of models have been proposed for the study of synchronous parallel computation. We review these models and study further some prototype problems. Within a spectrum of shared memory models, we show that loglog n is asymptotically optimal for n processors to merge two sorted lists containing n elements.

* University of Toronto
** Cornell University

This research was supported in part by ONR contract N00014-76-C-0018.

I Introduction: What is a reasonable model?

A number of relatively diverse problems are often referred to under the topic of "parallel computation". The viewpoint of this paper is that of a "tightly coupled", synchronized (by a global clock) collection of parallel processors, working together to solve a terminating computational problem. Such parallel processors already exist and are used to solve time consuming problems in a wide variety of areas including computational physics, weather forecasting, etc. The current state of hardware capabilities will facilitate the use of such parallel processors to many more applications as the speed and the number of processors that can be tightly coupled increases dramatically.

Within this viewpoint, Preparata and Viullemin [7] distinguish two broad categories. Namely, we can differentiate between those models that are based on a fixed connection network of processors and those that are based on the existence of global or shared memory. In the former case, we assume that only graph theoretically adjacent processors can communicate in a given step, and we usually assume that the network is reasonably sparse; as examples, consider the shuffle-exchange network (Stone [10]) and its development into the Ultracomputer of Schwartz [8], the array or mesh connected processors such as the Illiac IV, the cube-connected cycles of Preparata and Viullemin [7], or the more basic n-dimensional hypercube studied in Valiant and Brebner [12]. As examples of models based on shared memories, there are the PRAC of Angluin and Valiant [1], the PRAM of Fortune and Wyllie [2], the unnamed parallel model of Shiloach and Vishkin [9], and the SIMDAG of Goldschlager [3]. Essentially these models differ in whether or not they allow fetch and write conflicts, and if allowed, how write conflicts are resolved.

From a hardware point of view, fixed connection models seem more reasonable and, indeed, the global memory-processor interconnection would probably be realized in practice by a fixed connection network (see Schwartz [8]). Furthermore, for a number of important problems (eg, FFT, bitonic merge, etc.) either the shuffle-exchange or the cube connected cycles provide optimal hosts for well known algorithms. On the other hand, many problems require only infrequent and irregular processor communication, and in any case the shared memory framework seems to provide a more convenient environment for constructing algorithms. Finally, in defense of the PRAM, it is plausible to assume that some broadcast facilities could be made available.

The problem of sorting, and the related problem of routing are prototype problems, due both to their intrinsic significance and their role in processor communication. Since merging is a (the) key subroutine in many sorting strategies, we are interested in merging and sorting with respect to both the fixed connection and shared memory models. In a companion paper, we study the routing problem for fixed connection networks, such as the n-dimensional cube. For such a machine, the complexity of merging has been resolved by the fundamental log n algorithms of Batcher (see Knuth [5] for a discussion of odd-even and bitonic merge). The lower bound in this regard is immediate because log n is the graph theoretic diameter. In this paper, we concentrate on the complexity of merging (with application to sorting) on shared memory machines.

II A Hierarchy of Models

The shared memory models usually studied all possess a global memory, each cell of which can be read or written by any processor. For the purpose of constructing algorithms, one usually assumes a single instruction stream: that is, one program is executed by all processors. However, when the processor number itself is used to control the sequencing of steps, and some ability to synchronize control is introduced, then the effect is that of a multiple instruction stream. The processors are assumed to have some local memory and each processor can execute basic primitive operations such as $\leq, =, \neq$ comparisons and integer $+, -, \times, \div$ arithmetic operations in a single step. The following models have been considered:

1. PRAC (Angluin and Valiant) - Simultaneous read or write (of the same cell) is not allowed.

2. PRAM (Fortune and Wyllie) - Simultaneous fetches are allowed but no simultaneous writes.

3. WRAM - WRAM denotes a variety of models that allow simultaneous reads and (certain) writes, but differ in how such write conflicts are to be resolved.

   a. (Shiloach and Vishkin) a simultaneous write is allowed only if all processors are trying to write the same thing, otherwise the computation is not legal.

   b. An arbitrary processor is allowed to write

   c. (Goldschlager) the lowest numbered processor is allowed to write.

Other variants are clearly possible. We are concerned with the merging and sorting problems of elements from an arbitrary linear order (i.e. the schematic or structured approach). In this context, a "most powerful" parallel model (analagous to the comparison tree for sequential computation) has been studied by Valiant. The _parallel computation tree_ idealizes k-processor parallelism by a 3^k-tree where each node is labelled by a set of k {<,=,>} comparisons and the branches are labelled by each of the 3^k possible outcomes. It should be clear that for the problems of concern, parallel computation trees can simulate any reasonable parallel model, and in particular, can simulate all of the aforementioned shared memory models.

Let M denote any of these models. We will be concerned with T^M_{merge} (n,m,p) and T^M_{sort} (n,p), the minimum number of parallel steps to merge two sorted lists of n and m elements (respectively, to sort n arbitrary elements) using p processors. Typically, n=m, and p=O(n) or $O(n \log^\alpha n)$. Clearly, for any problem we have

$$T^{PRAC} \geq T^{PRAM} \geq T^{WRAM}.$$

Our main contribution is to establish the following two theorems:

Theorem 1: Let M denote the parallel computation tree model. Then $T^M_{\text{merge}}(n,n,n^\alpha) = \Omega(\text{loglog } n)$ for $\alpha<2$.

Theorem 2: $T^{PRAM}_{\text{merge}}(n,n,n) = O(\text{loglog } n)$.

We use Valiant's algorithm, which already establishes the bound for the parallel comparison tree, but following Valiant [11], Preparata [6] and Shiloach and Vishkin [9], remark that a "processor allocation" problem must be solved to realize Valiant's algorithm on the PRAM model.

Hence, the problem of merging is now resolved on all of the above shared memory models except the PRAC, for which we cannot improve on the log n upper bound of the Batcher merge. For the PRAC, it is not difficult to show that $\Omega(\sqrt{\log n})$ is a lower bound for insertion (and hence merging); indeed we conjecture that insertion requires $\Omega(\log n)$ on a PRAC.

With regard to sorting, we have the following direct corollaries:

<u>Corollary 1</u>: $T^{PRAM}(n,n) = O(\log n \operatorname{loglog} n)$.

<u>Corollary 2</u>: $T^{PRAM}(n,n \log n) = O(\log n)$.

Clearly, Corollary 1 follows from a standard merge sort, whereas Corollary 2 is a restatement of Preparata's [6] result, which can now be stated for PRAM's using Theorem 2. Corollaries 1 and 2 should be compared with the Shiloach and Vishkin upper bound of $O(\log^2 \frac{n}{\log(p/n)} + \log n)$ for sorting on their version of a WRAM with p processors. With regard to lower bounds for sorting, Haagvist and Hell [4] prove that in terms of the parallel computation tree, time less than or equal to k implies $\Omega(n^{1+1/k})$ processors are required (and this is essentially sufficient). It follows, that for the tree model and any of the RAM models, $\Omega(\log n / \log\log n)$ is a lower bound for sorting with $\Omega(n \log^{\alpha} n)$ processors. For O(n) processors, $\Omega(\log n)$ is a trivial lower bound resulting from the sequential lower bound of $\Omega(n \log n)$. Among the open questions for parallel sorting are the following: the number of processors for $O(\log n)$ time sorting on a PRAC (Preparata [6] achieves $O(k \log n)$ time with $n^{1+1/k}$ processors); whether it is possible to sort in time $o(\log n)$, and in particular in time O(1), on a PRAC or PRAM; whether it is possible to sort in time O(1) on a WRAM using only polynomial in n number of processors.

III A Ω(loglog n) lower bound for merging on Valiant's model

Sequentially merging two lists of length n can be accomplished with 2n-1 comparisons and this is provably optimal. Since only 2n-1 comparisons are necessary to merge two such lists, conceivably in a parallel model they could be merged in time O(1) with n processors. However, we shall show that this is not possible. Even allowing n^α for $\alpha<1$ comparisons per step, a depth of loglog n is needed.

Consider the process of merging two sorted lists $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ with n processors. At the first step at most n comparisons can be made. Partition each list into $2\sqrt{n}$ blocks of length $\frac{1}{2}\sqrt{n}$. Form pairs of blocks, one from each list. There are 4n such pairs of blocks. Clearly there must be 3n pairs (A_i,B_j) of blocks such that no element from the block A_i is compared with any element from the block B_j. We shall show that we can select $\frac{1}{2}\sqrt{n}$ pairs of blocks

$$(A_{i_1},B_{j_1}),(A_{i_2},B_{j_2}), \ldots, (A_{i_{\frac{1}{2}\sqrt{n}}},B_{j_{\frac{1}{2}\sqrt{n}}})$$

such that $i_l<i_{l+1}$ and $j_l<j_{l+1}$ for $1\le l<\frac{1}{2}\sqrt{n}$. If the total order is such that all elements in $A_{i_l} \cup B_{j_l}$ are less than any element in $A_{i_{l+1}} \cup B_{j_{l+1}}$, $1\le l<\frac{1}{2}\sqrt{n}$, then after the first stage we are faced with $\frac{1}{2}\sqrt{n}$ subproblems each of size $\frac{1}{2}\sqrt{n}$.

At the second stage the n processors are partitioned somehow among the $\frac{1}{2}\sqrt{n}$ subproblems. However this is done, at least one half of the subproblems have assigned to them fewer than twice the average available number of processors per subproblem. Thus there are $\frac{1}{4}\sqrt{n}$ subproblems with at most $4\sqrt{n}$ processors per problem. Intuitively this argument suggests that at each stage the size of subproblem goes down by a square root and hence loglog n time is necessary. These ideas will be made precise in the following lemmas.

In what follows let $G = (A \cup B, E)$ be a bipartite graph with $E \subset A \times B$. Further let $A_1, A_2, \cdots$ and $B_1, B_2, \cdots$ be fixed orderings of the vertices in A and B, respectively. A matching is said to be <u>compatible</u> if for each pair of edges (A_g, B_h) and (A_i, B_j) in the matching $i<h$ if and only if $j<g$.

<u>Lemma</u>: Let $G=(A \cup B, E)$ be a bipartite graph with $A = A_1, A_2, \cdots, A_{2k}$ and $B = B_1, B_2, \cdots, B_{2k}$ and let $E \subset A \times B$ have $3k^2$ edges. Then G has a compatible matching of cardinality at least k.

<u>Proof</u>: Partition the edges into 2k-1 blocks as follows. For $-k<b<k$ we have a block consisting of edges $\{(A_i, B_{i+b}) | 1 \le i \le 2k \text{ and } 1 \le i+b \le 2k\}$. In addition we have one block consisting of all other edges. At most $2k^2$ edges fall into the block of other edges. Thus at least k^2 edges must be partitioned into 2k-1 blocks. Hence at least one block must have at least k edges and these edges form a compatible matching.

<u>Lemma</u>: Let $T(s,c)$ be the time necessary to solve k, $k \ge 1$, problems of size s with cks processors. Then $T(s,c)$ is $\Omega(\frac{\log \log sc}{\log c})$.

<u>Proof</u>: On the average we can assign cs processors to each problem. At least one half of the problems can have no more than twice this number of processors assigned to them. That is, at least k/2 problems have at most 2cs processors.

Consider applying 2cs processors to a problem of size s. This means that in the first step we can make at most 2cs comparisons. Partition the lists into $2\sqrt{2cs}$ blocks each of size $\frac{1}{2}\sqrt{\frac{s}{2c}}$. There are 8cs pairs of blocks. Thus there must be 6cs pairs of blocks with no comparisons between elements of the blocks in a pair. Construct a bipartite graph whose vertices are the blocks from the two lists with an edge between two blocks if there are no comparisons between elements of the two blocks. Clearly there are 6cs edges and thus by the previous lemma there is a compatible match of size at least $\frac{1}{2}\sqrt{2cs}$. This means that there are at least $\frac{1}{2}\sqrt{2cs}$ problems each of size at least $\frac{1}{2}\sqrt{\frac{s}{2c}}$ that we must still solve. Thus $T(s,c) \geq 1+T(\frac{1}{2}\sqrt{\frac{s}{2c}},4c)$.

We show by induction on s, that

$$T(s,c) \geq d\log\frac{\log sc}{\log c}$$

for some sufficiently small d.

$$\begin{aligned} T(s,c) &\geq 1+d\log\frac{\log\frac{1}{2}\sqrt{\frac{s}{2c}}4c}{\log 4c} \\ &\geq 1+d\log\left(\frac{\log sc}{4\log c}\right) \\ &\geq 1+d\log\frac{\log sc}{\log c}-d\log 4 \\ &\geq d\log\frac{\log sc}{\log c} \end{aligned}$$

provided $d<\frac{1}{2}$. Observe that $\log\frac{\log sc}{\log c}$ is $\Omega(\log\log s - \log\log c)$ which matches Valiant's upper bound of $2(\log\log s - \log\log c)$.

IV An O(loglog n) upper bound for merging on a PRAM

We recall Valiant's (n,m) merging algorithm which merges X and Y with $\#X=n$, $\#Y=m$, $n\leq m$ using $\sqrt{nm}$ processors. Our goal is to implement Valiant's algorithm on a PRAM. The algorithm (taken verbatim from Valiant [11]) proceeds as follows:

(a) Mark the elements of X that are subscripted by $i \cdot \lceil\sqrt{n}\rceil$ and those of Y subscripted by $i \cdot \lceil\sqrt{m}\rceil$ for $i = 1,2, \ldots$. There are at most $\lfloor\sqrt{n}\rfloor$ and $\lfloor\sqrt{m}\rfloor$ of these, respectively. The sublists between successive marked elements and after the last marked element in each list we call <u>segments</u>.

(b) Compare each marked element of X with each marked element of Y. This requires no more than $\lfloor\sqrt{nm}\rfloor$ comparisons and can be done in unit time.

(c) The comparisons of (b) will decide for each marked element the segment of the other list into which it needs to be inserted. Now compare each marked element of X with every element of the segment of Y that has thus been found for it. This requires at most

$$\lfloor\sqrt{n}\rfloor \cdot (\lceil\sqrt{m}\rceil - 1) < \lfloor\sqrt{nm}\rfloor$$

comparisons altogether and can also be done in unit time.

On the completion of (a), (b) and (c) we can store each $X_{i\lceil\sqrt{n}\rceil}$ in its appropriate place in the output Z. It then remains to merge the disjoint pairs of sublists $(X_0,Y_0),(X_1,Y_1), \ldots$ where the X_i and Y_i are segments of X and Y respectively. Whereas Chauchy's inequality guarantees that there will be enough processors to carry out these independent merges by simultaneous recursive calls of the algorithm, it is not clear how to inform each processor to which (X_i,Y_i) subprogram (and in what capacity) it will be assigned. This is the main concern in what Shiloach and Vishkin [9] refer to as the <u>processor allocation problem</u>.

We desire a recursive procedure (in fact, a macro might be more appropriate) MERGE(i,n_i,j,m_i,k) which merges $X_i,X_{i+1},\ldots,X_{i+n_i-1}$ and $Y_j,\ldots,Y_{j+m_i-1}$ into $Z_{i+j-1},\ldots,Z_{i+j+n_i+m_i-2}$ using at most $\sqrt{n_im_i}$ processors beginning at processor number p_k. Such a merge will be simultaneously invoked by processors $p_k,p_{k+1},\ldots,p_{k+\sqrt{n_im_i}-1}$. The initial call is MERGE(1,n,1,m,1). As we enter this subroutine, a processor p_k will know from i,j,n_i,m_i, and k, the (relative) role it plays in parts (a), (b) and (c) of Valiant's algorithm. For example, say $n_i \le m_i$ and let

$$l = k + i'\lceil\sqrt{n_i}\rceil + j' \qquad 0\le i'\le\lfloor\sqrt{n_i}\rfloor - 1$$
$$0\le j'\le\lfloor\sqrt{m_i}\rfloor - 1$$

then in step (a), processor p_l *compares* $X_{i'\cdot\sqrt{n_i}+i}$ *and* $Y_{j'\cdot\sqrt{m_i}+j}$.

We will now indicate how processors reassign themselves before recursively invoking the merge routine. For simplicity, assume that we have just completed steps (a),(b),(c) of MERGE(1,n,1,m,1). We can assume that we have determined for each i, $0\le i\le\lfloor\sqrt{n}\rfloor-1$ that $Y_{j_i\lceil\sqrt{m}\rceil} < X_i \le Y_{(j_{i+1})\lceil\sqrt{m}\rceil}$ and that we have constructed a table J

0	j_i	j_2	......	$j_{\sqrt{n}-1}$	m

accessible by all $\sqrt{nm}$ processors. A given processor p must determine its role in the next iteration of the algorithm.

<u>Lemma</u>: Suppose $(X_0,Y_0),\ldots,(X_{r-1},Y_{r-1})$ have been assigned processors, and $X_{r-1} = (\ldots,X_{r\sqrt{n}-1})$ and $Y_{r-1} = (\ldots,y_f)$. There exists a

function ϕ such that no more than $\phi(r,n,f)$ processors have been assigned. Indeed, $\phi(r,n,f) = r \cdot n^{1/4} \cdot \sqrt{\frac{f}{r}}$. (The bound $rn^{1/4}\sqrt{\frac{f}{r}}$ is achieved by considering the case $Y_i = f/r$ for all $i \le r$.)

Proof: *Cauchy's inequality will do the job here.*

The impact of this Lemma is that we can safely assign processors $P_{\phi(r,n,f)+1}, \ldots, P_{\phi(r+1,n,f)}$ to (X_r,Y_r) It remains for each processor to know to which (X_k,Y_k) it will be assigned. Indeed, once a processor knows to which (X_k,Y_k) it has been assigned, then it can obtain all the information it will need to invoke MERGE from the table J and the ϕ function; namely, X_k starts at $k\lceil\sqrt{n}\rceil$ and has length $\lfloor\sqrt{n}\rfloor - 1$, Y_k starts at Y_{j_k} and has length $j_{k+1}-j_k+1$, and the processors assigned to this task began at $P_{\phi(k,n,j_{k-1}+1)}$.

The actual assignment of a processor to a (X_k,Y_k) subproblem proceeds in two stages (note that we cannot simply do a sequential binary search in J because this would require $\log\sqrt{m}$ steps):

Stage 1)

Processors are assigned for those (X_k,Y_k) with $\#Y_k \le \sqrt{n}$ (and hence no more than $\sqrt{n}$ processors need be assigned to this task since $\#X_i = \sqrt{n} - 1$ for all i.

Stage 2)

Processors are assigned to the remaining (X_k,Y_k).

Stage 1: For each k, $0 \le k \le \sqrt{m} - 1$, we assign $\sqrt{n}$ processors to look at both the k and the k+1st entry of the table J. If $j_{k+1}-j_k \le \sqrt{n}$, then these $\sqrt{n}$ processors inform (by posting the information in an appropriate place of global memory) processors numbered $\phi(k,n,j_{k-1})+1, \ldots, \phi(k+1,n,j_k)$ that they are assigned to (X_k,Y_k). We then wait until the completion of Stage 2 before invoking merge on (X_k,Y_k) since all $\sqrt{nm}$ processors are needed for Stage 2.

<u>Stage 2</u>: The processors are divided into $\sqrt{m}$ blocks, each block containing $\sqrt{n}$ processors. Each of the $\sqrt{n}$ processors in a block are trying to determine to which X_k,Y_k these $\sqrt{n}$ processors will be assigned. Let p_{j_l} be the first processor of block l. The kth processor of block l looks at the j_k and j_{k+1} in table J and determines (via the function ϕ) whether or not processor p_{j_l} would be assigned to this subproblem. Now each processor p in the lth block can determine (again via table J and ϕ) which of the following hold:

i) p is assigned to (X_k,Y_k), the subproblem assigned to p_{j_l}

ii) p is assigned to $(X_{k'},Y_{k'})$, the subproblem assigned to $p_{j_{l+1}}$

iii) p has already been assigned in Stage 1.

We claim that if neither i) and ii) hold, then iii) must hold since clearly less than $\sqrt{n}$ processors have been assigned to the same task as p.

□

V <u>Application to sorting and open problems</u>

Preparata [6] derives a set of parallel sorting algorithms, all based on what Knuth [5] calls enumeration sorting. The "count acquisition stage" is often accomplished by merging. Using Batcher's merge, sorting can then be performed in O(k log n) steps on a PRAC using $n^{1+1/k}$ processors. More to the thrust of this paper, Preparata shows how Valiant's merge can be used to derive a O(log n) time sort using n log n processors. Since we have shown how to implement Valiant's merge on a PRAM, Preparata's bounds will now be applicable to the PRAM model. It is also clear that with only n processors, a merge sort will take O(log n loglog n) time.

A number of open problems concerning time vs number of processors are readily suggested by the above comments. We can classify two sets of questions:

1) The number of processors required for an O(log n) time sort on the various models, the present upper bounds being $n^{1+1/k}$ (PRAC), n log n (PRAM, WRAM, parallel computation tree). In all cases, n processors is an obvious lower bound.

2) For what models is it possible to sort in o(log n) time and, if possible, how many processors are required? In particular, in O(1) time, sorting can be done using $O(2^n)$ processors on a WRAM or in constant time k using $n^{1+1/k}$ processors on a parallel computation tree. We do not know if such a fast sort is possible for the PRAC or PRAM.

References

[1] Angluin, D., and L.G. Valiant. Fast probabilistic algorithms for Hamilton circuits and matchings. Proceedings 9th Annual ACM Symposium on Theory of Computing, Boulder, Colorado (1977), 30-41.

[2] Fortune, S., and J. Wyllie. Parallelism in random access machines. Proceedings 10th Annual ACM Symposium on Theory of Computing, San Diego, California (1978), 114-118.

[3] Goldschlager, L. A unified approach to models of synchronous parallel machines. Proceedings 10th Annual ACM Symposium on Theory of Computing, San Diego, California (1978), 89-94.

[4] Haagvist, R., and P. Hell. Parallel sorting with constant time for comparisons. SIAM J. on Computing 10:3 (1981), 465-472.

[5] Knuth, D.E. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1972.

[6] Preparata, F.P. New parallel-sorting schemes. IEEE Transactions on Computers C27:7 (1978), 669-673.

[7] Preparata, F.P., and J. Viullemin. The cube-connected cycles. Proceedings 20th Symposium on Foundations of Computer Science (1979), 140-147.

[8] Schwartz, J.T. Ultracomputers. ACM TOPLAS 2 (1980), 484-521.

[9] Shiloach, Y., and U. Vishkin. Finding the maximum, merging and sorting in a parallel computation model. Department of Computer Science, Technion Israel, TR173, March 1980.

[10] Stone, H. Parallel processing with the perfect shuffle. IEEE Transactions on Computers C20:2 (1971), 153-161.

[11] Valiant, L.G. Parallelism in comparison problems. SIAM J. on Computing 4 (1975), 348-355.

[12] Valiant, L.G., and G.J. Brebner. Universal schemes for parallel computation. Proceedings 13th Annual ACM Symposium on Theory of Computation, Milwaukee, Wisconsin (1981), 263-277.

UNCLASSIFIED
Security Classification

AD-A106 189

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Science Department Cornell University Ithaca, New York 14853	2a. REPORT SECURITY CLASSIFICATION 2b. GROUP

3. REPORT TITLE

MERGING ON PARALLEL MODELS OF COMPUTATION

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report

5. AUTHOR(S) (First name, middle initial, last name)
John E. Hopcroft
Allan Borodin

6. REPORT DATE October 1981	7a. TOTAL NO. OF PAGES 15	7b. NO. OF REFS 12
8a. CONTRACT OR GRANT NO. N00014-76-C-0018 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) TR81-472 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

Distribution of manuscript is unlimted.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

A variety of models have been proposed for the study of synchronous parallel computation. We review these models and study further some prototype problems. Within a spectrum of shared memory models, we show that loglog n is asymptotically optimal for n processors to merge two sorted lists containing n elements.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6801

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
PARALLEL MERGING SORTING NETWORK ROUTING						

DATE
ILME